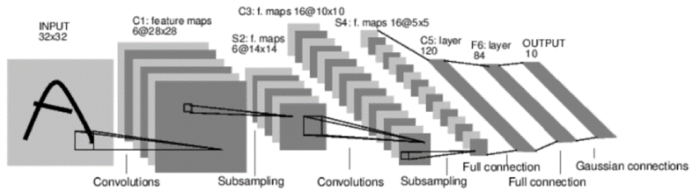
"Graph CNNs"

Ray Ptucha

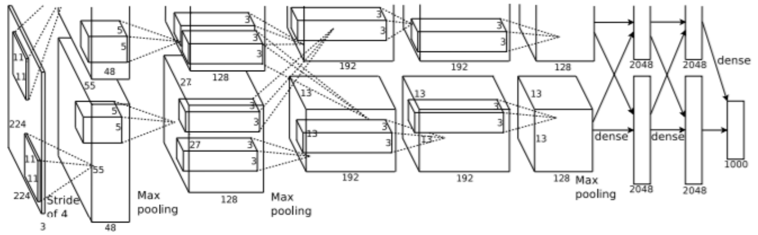
Rochester Institute of Technology

Nov, 28 2018

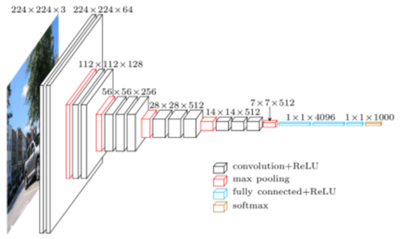
Convolution Neural Networks (CNNs) are awesome- they have transformed the pattern recognition community!



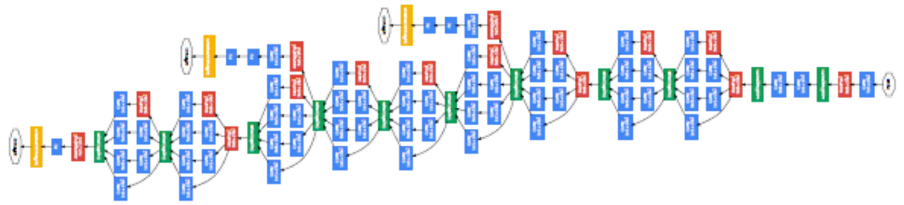
LeNet-5, LeCun 1989



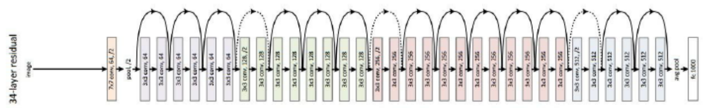
AlexNet, Krizhevsky 2012



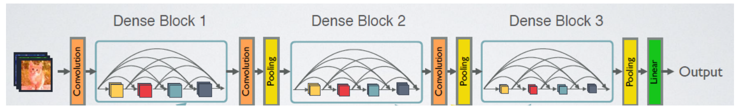
VGGNet, Simonyan 2014



GoogLeNet (Inception), Szegedy 2014



ResNet, He 2015



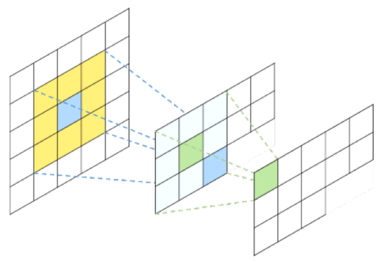
DenseNet, Huang 2017

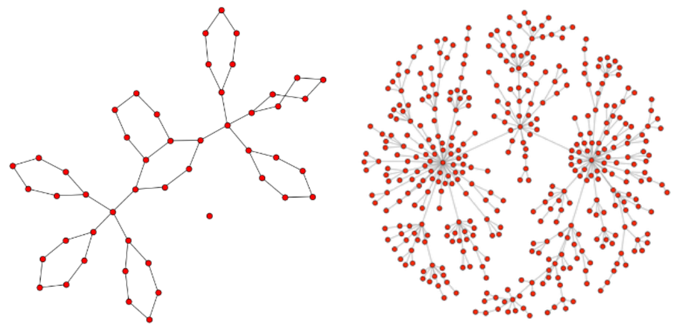
But, the vast majority of the world's problems can't be described by gridded structures such as images. Have you ever tried to do a CNN on a graph?

Images

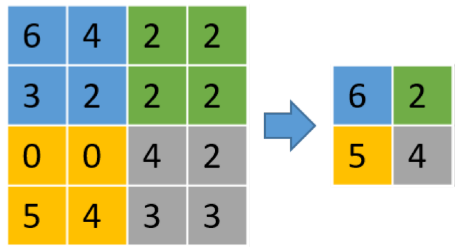
Graphs

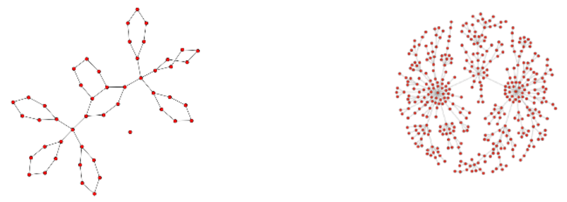
- Convolution



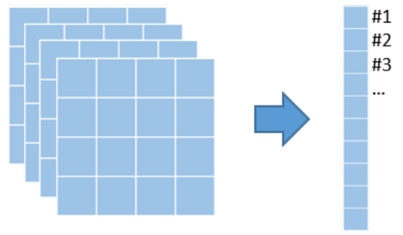


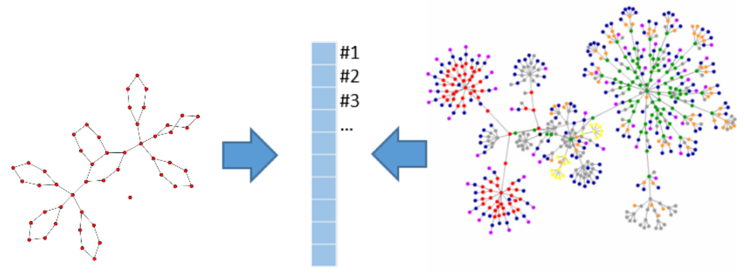
- Pooling



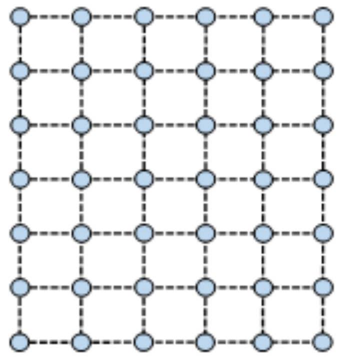


- Classification

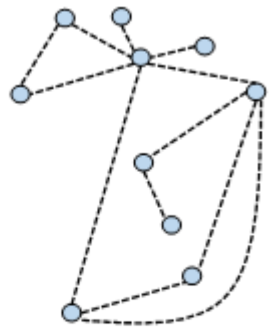




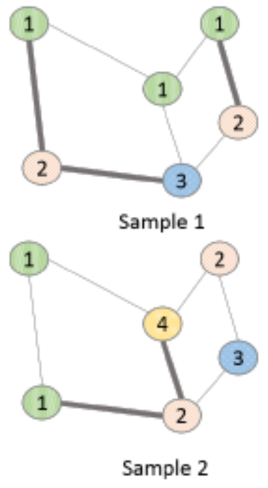
GraphCNN affords the wonderful CNN benefits to non-gridded problems such as trade, security, protein structures, weather, brain scans, etc.



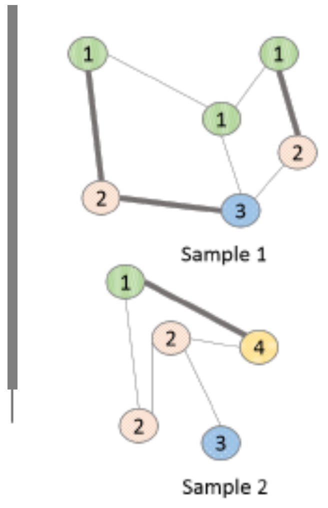
Gridded



Non-Gridded

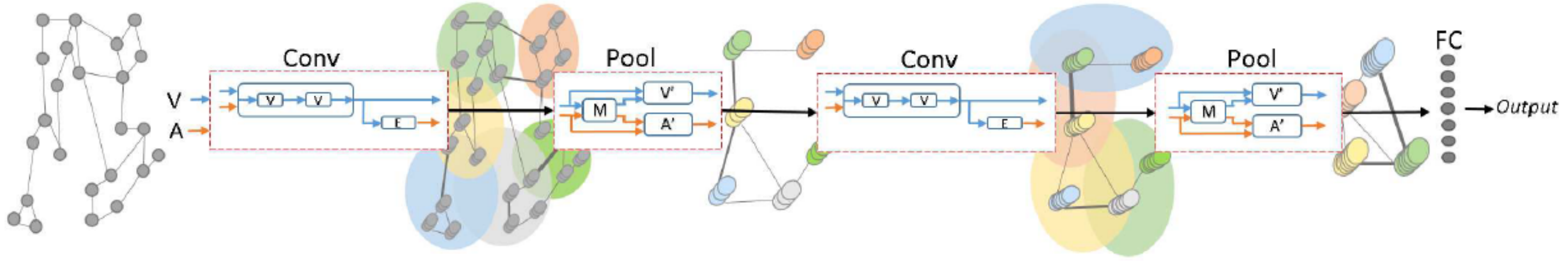


Homogeneous

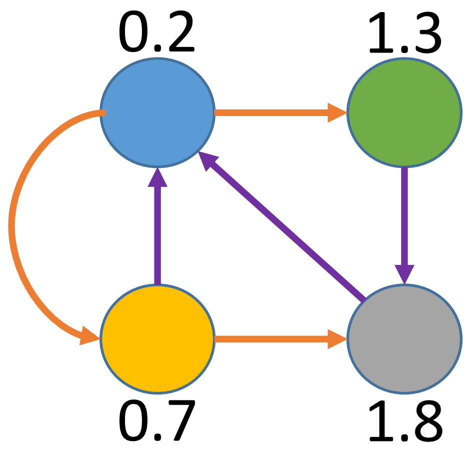


Heterogeneous

Graph CNNs introduced by Lab:



Graph-CNN - Convolution



Adjacency Matrix

Incoming Purple Connections

A

🟦	🟩	🟨	⬜
0	0	1	1
0	0	0	0
0	0	0	0
0	1	0	0

Incoming Orange Connections

A

🟦	🟩	🟨	⬜
0	0	0	0
1	0	0	0
1	0	0	0
0	0	1	0

Vertices

V

0.2
1.3
0.7
1.8

Can have multiple connection types (A can be a tensor) and can have multiple features (V can be a tensor)

Graph-CNN - Convolution

A

0	0	1	1
0	0	0	0
0	0	0	0
0	1	0	0

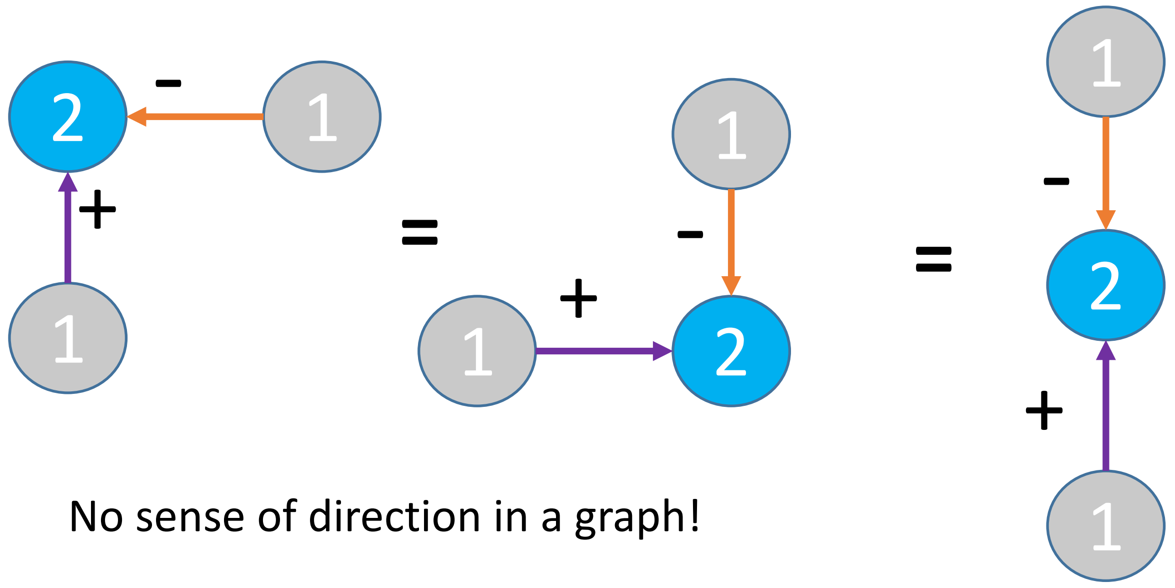
A

0	0	0	0
1	0	0	0
1	0	0	0
0	0	1	0

V

0.2
1.3
0.7
1.8

Replace each vertex with 2× itself + 1× incoming purple - 1× incoming orange



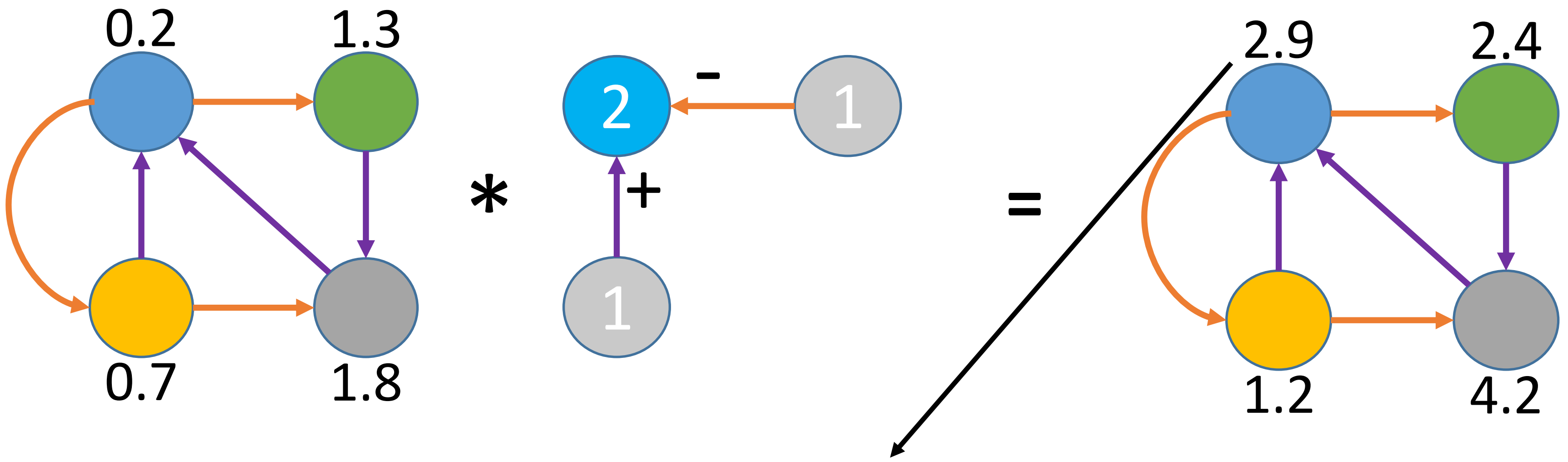
No sense of direction in a graph!

Graph-CNN - Convolution

$$\begin{array}{c} A \\ \begin{bmatrix} 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \end{array} \quad \begin{array}{c} A \\ \begin{bmatrix} 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \end{array} \quad \begin{array}{c} V \\ \begin{bmatrix} 0.2 \\ 1.3 \\ 0.7 \\ 1.8 \end{bmatrix} \end{array}$$

In Graph CNN, we will learn many such filters (like the [2 1 -1]) per adjacency matrix.

Replace each vertex with 2× itself + 1× incoming purple - 1× incoming orange



2× (0.2) + 1× (0.7+1.8) - 1× (0)

Graph-CNN - Convolution

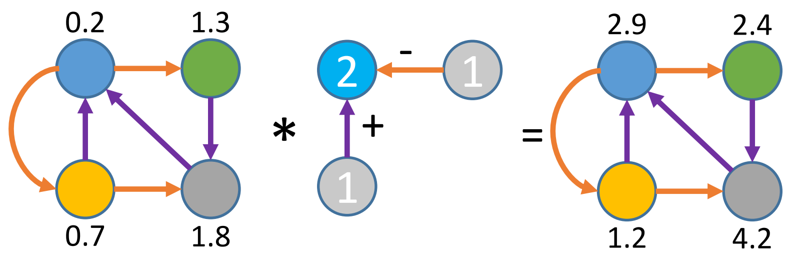
A

0	0	1	1
0	0	0	0
0	0	0	0
0	1	0	0

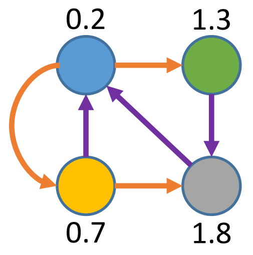
A

0	0	0	0
1	0	0	0
1	0	0	0
0	0	1	0

V
0.2
1.3
0.7
1.8



Update V’s for incoming A’s



V	AV	AV
0.2	2.5	0
1.3	0	0.2
0.7	0	0.2
1.8	1.3	0.7

$$N^{l}_{i,aC+b} = A_{i,a,j} V^{l}_{j,b}$$

Matrix Multiplication

V	AV	AV
0.2	2.5	0
1.3	0	0.2
0.7	0	0.2
1.8	1.3	0.7

X

2
1
-1

=

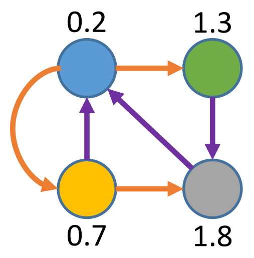
2.9
2.4
1.2
4.2

$$V^{l}_{i,k} = f\left(N^{l-1}_{i,d} W_{d,k}\right)$$

Graph-CNN - Convolution

- We are learning the weights of the filters.
- We don't care how many vertices!
- Can learn several sets of weights, one for each filter.

Update V's for incoming A's



V	AV	AV
0.2	2.5	0
1.3	0	0.2
0.7	0	0.2
1.8	1.3	0.7

$$N^{l}_{i,aC+b} = A_{i,a,j} V^{l}_{j,b}$$

Matrix Multiplication

$$\begin{array}{ccc} V & AV & AV \\ 0.2 & 2.5 & 0 \\ 1.3 & 0 & 0.2 \\ 0.7 & 0 & 0.2 \\ 1.8 & 1.3 & 0.7 \end{array} \; X \; \begin{bmatrix} 2 \\ 1 \\ -1 \end{bmatrix} = \begin{bmatrix} 2.9 \\ 2.4 \\ 1.2 \\ 4.2 \end{bmatrix}$$

$$V^{l}_{i,k} = f\left(N^{l-1}_{i,d} W_{d,k}\right)$$

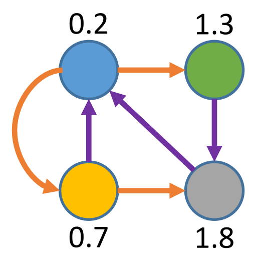
CNN – Fully Connected Layer

$$\begin{bmatrix} 6 & 2 \\ 5 & 4 \end{bmatrix} \circ \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} = \begin{bmatrix} 3 \end{bmatrix}$$

- Matrix multiplication.
  - Parameters are learned.
- Requires fixed input shape, size, and order.
- Obtains representation vector.

Graph-CNN – Graph Representation Vector

Turn arbitrary # vertices into a single vertex



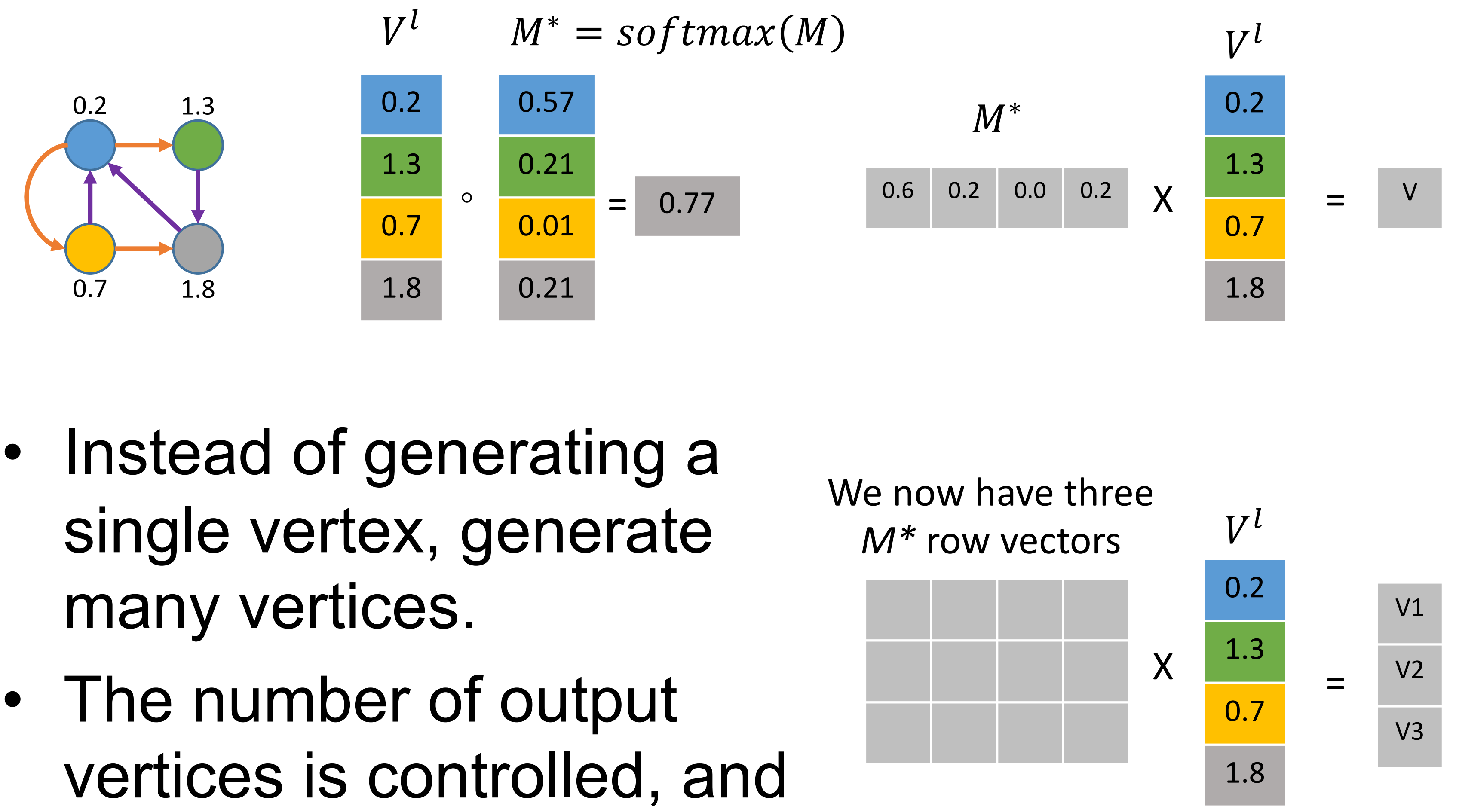
$$\begin{array}{ccccc} V & & M & & V^T M \\ \begin{bmatrix}0.2\\1.3\\0.7\\1.8\end{bmatrix} & \circ & \begin{bmatrix}2\\1\\-2\\1\end{bmatrix} & = & \boxed{2.1} \end{array}$$

$$\begin{array}{ccccc} V & & M^* = softmax(M) & & V^T M^* \\ \begin{bmatrix}0.2\\1.3\\0.7\\1.8\end{bmatrix} & \circ & \begin{bmatrix}0.57\\0.21\\0.01\\0.21\end{bmatrix} & = & \boxed{0.77} \end{array}$$

```
x = [2 1 -2 1]
M* = exp(x)./sum(exp(x))
[0.2 1.3 0.7 1.8]*M*'
```

- Graphs can have varying vertices, but we often need fixed nodes for say a final classification task.
- Define a soft attention applied to vertices of graph- learn *M*, a linear combination of all vertices.
- This reduces all vertices to a single vertex.
- A softmax is applied to *M* before computing linear combination, this ensures the sum of the weights=1.

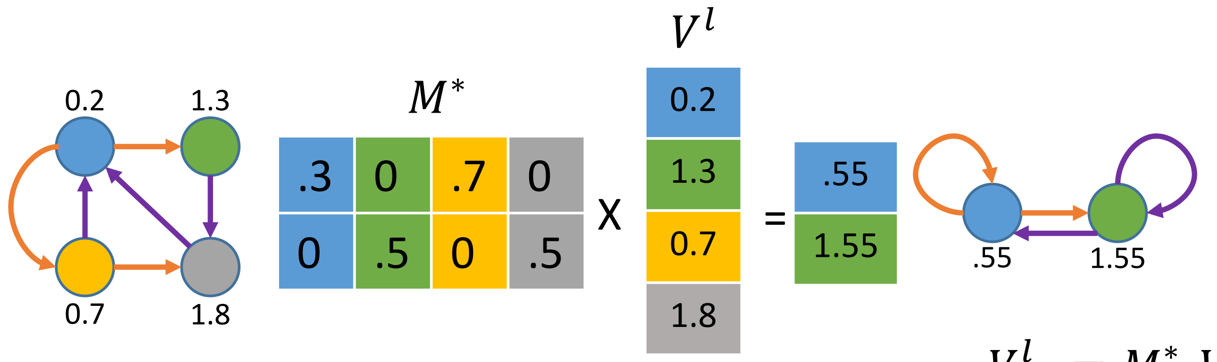
Graph-CNN – Embed Pooling



- Instead of generating a single vertex, generate many vertices.
- The number of output vertices is controlled, and can simplify the model.

If FC layer has ten connections, learn ten M^* row vectors!

Graph-CNN – Embed Pooling



$$V^l_{i,d} = M^*_{i,j} V^{l-1}_{j,d}$$

$$A^l_{i,a,j} = M^*_{i,x} A^{l-1}_{x,a,y} M^*_{j,y}$$

- Many Graph Representation Vectors combined.
- Output Adjacency matrix calculated accordingly.
- Fixed number of output vertices.
- Independent of number of input vertices.
- Results in fully connected graph, including self connections.

Graph-CNN – Applications

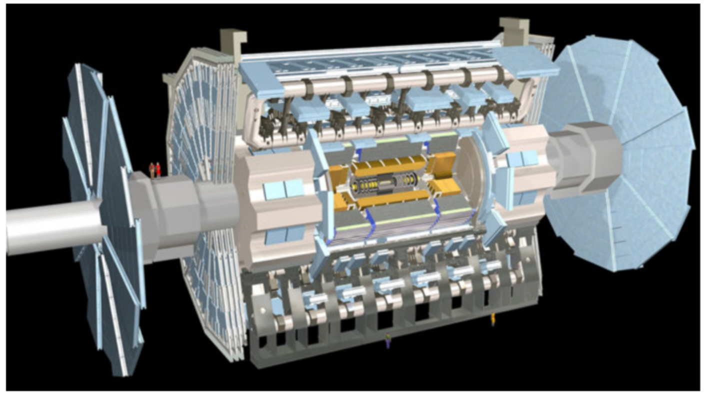
- Used for protein and chemical structures.
- Used for (LiDAR) point cloud object identification and point labeling.
- Used for document citation.
- Used for fMRI brain scan processing.
- Currently modifying for cardiac electrophysiology.

- Have extended TensorFow sparse library for large graphs.
- Have built neuroevolution machine for metalearning.

Latest news: Miguel Dominguez spent several days at Argonne Leadership Computing, Chicago

- Using GraphCNN to estimate energy signatures in their 5-story tall 3D particle detector, part of the ATLAS experiment at CERN's Large Hadron Collider.

- Due to the heterogeneity of their detectors, gridded CNNs don't work but graph CNNs do!



Thank you!!

Ray Ptucha
rwpeec@rit.edu

https://www.rit.edu/mil